RasPi

DESIGN
BUILD
CODE

50

Get hands-on with your Raspberry Pi

+ SEND SMS FROM YOUR PI

CREATE A RETRO SMARTPHONE

Plus Make a time-lapse camera trigger

Welcome

Upcycling is very much on trend and there's all manner of media dedicated to the art transforming useless or unwanted products into all-new objects of desire. As you might expect, a cutting-edge technology like Raspberry Pi is on the very crest of this wave, putting creative reuse at the heart of its myriad uses. In the past we've transformed old console controllers, old cameras and even old lamps into amazing new gadgets. This issue we've another great project that shows you how the head of ITC at a school in North Yorkshire, UK, transformed an old-school rotary telephone into a state-of-the-art smartphone. An amazing feat considering some his students didn't even know how to use one! Swipe left to find out more…

Get inspired

Discover the RasPi community's best projects

Expert advice

Got a question? Get in touch and we'll give you a hand

Easy-to-follow guides

Learn to make and code gadgets with Raspberry Pi

Editor

From the makers of

LinuxUser & Developer

Join the conversation at…

@linuxusermag | Linux User & Developer | linuxuser@futurenet.com

Contents

Create a gaming controller prototype for the Pi

Using some cheap and readily available parts we're going to build a working video games controller

The Raspberry Pi is such a versatile little computer. Having access to the GPIO pins enables us to interface directly with any number of hardware peripherals, and in this case, create our own.

Using some basic bits of tech, including a breadboard, some jumper wires and tactile push buttons, we're going to build our own prototype video games controller that'll work with third-party Pi games, but could also come in handy if you're coding games of your own. Our prototype won't be pretty, but there's nothing to say you couldn't 3D-print a case and turn the controller into a legitimate-looking gadget.

The beauty of a project like this is that you don't need to be an electrical engineer; in fact, you don't need to know much more than the basics of an electronic circuit, because all the thinking is done with code in Python. So grab a few wires, some buttons and a breadboard, boot up the Pi and let's jump into Python IDLE.

THE PROJECT ESSENTIALS

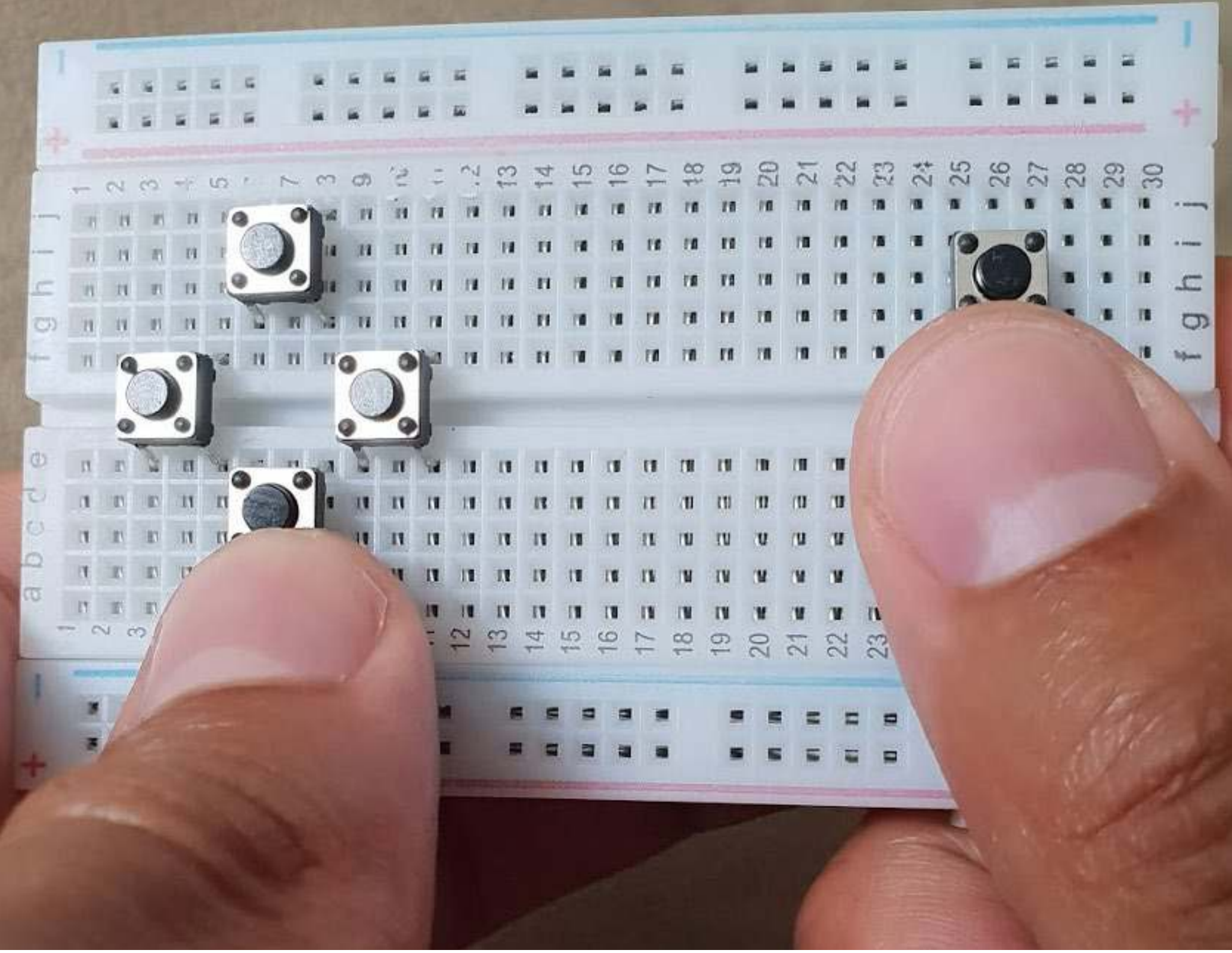
Raspberry Pi

Breadboard

Jumper wires

Tactile Push Buttons

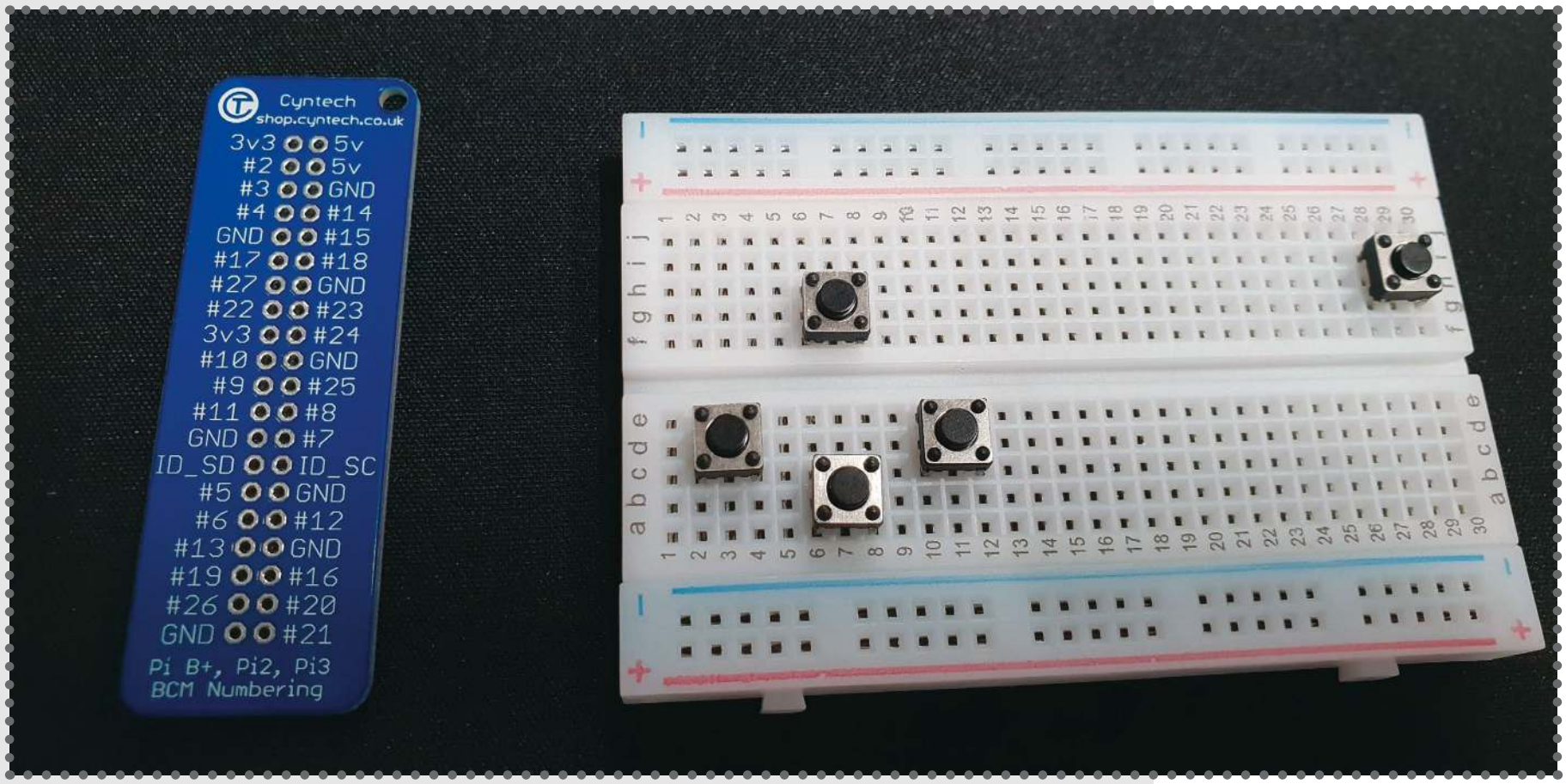
Pinout chart
https://pinout.xyz



01 Position buttons

You'll need to decide how many buttons your controller is going to have. We're starting with five for simplicity: up, down, left, right and select – the latter being the default action button used for everything from starting the game to making a character jump. If you're building your controller to work with a particular game you may want to consider your options.

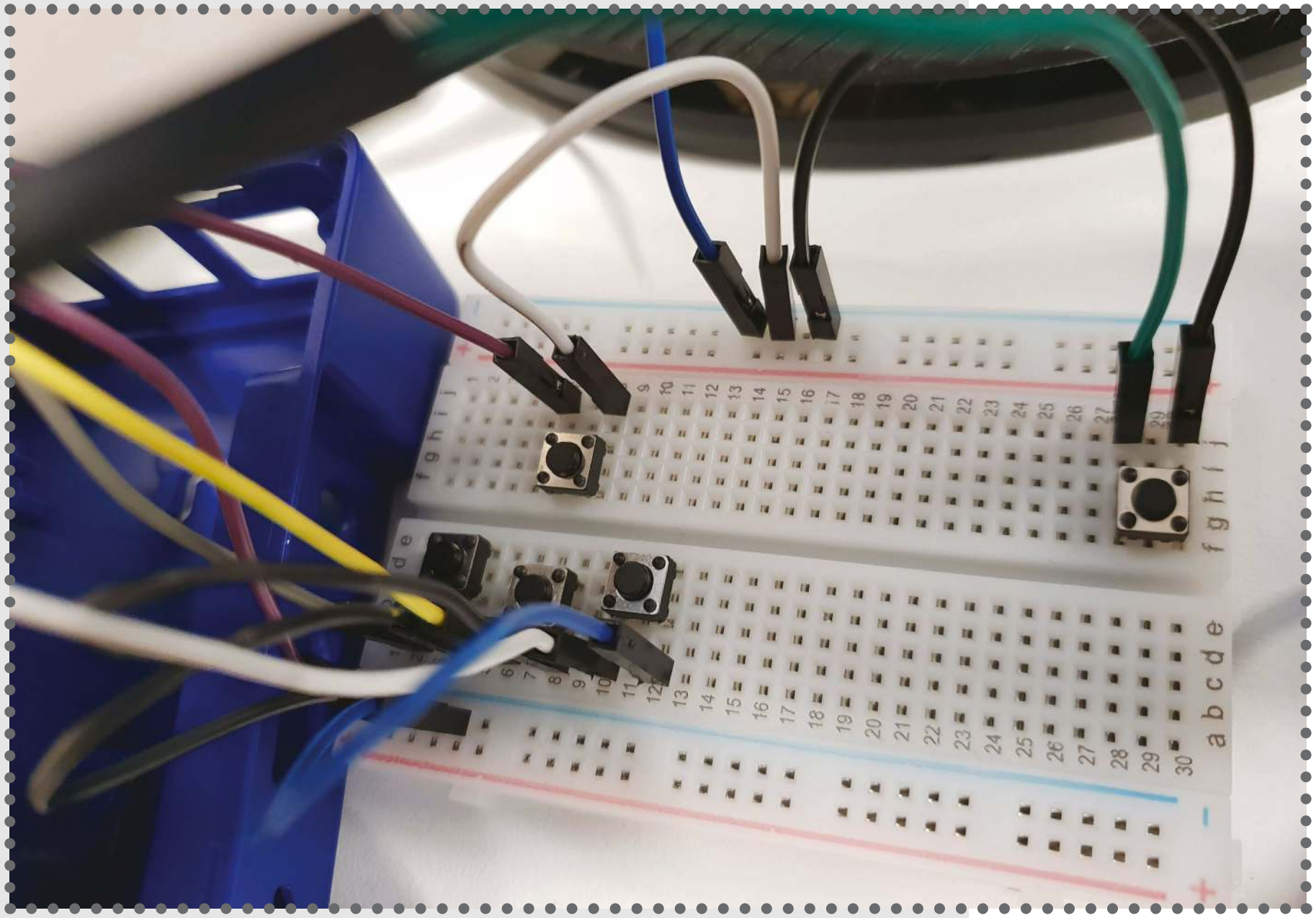
Place the buttons around the breadboard, leaving enough space to add two jumper wires per button: one for power and one for ground.



02 Connect the buttons

The next part can get a little messy. We need to connect the left side of each button to a GPIO pin on the Raspberry Pi (use the pinout chart to help with this). The GPIO pin will provide power and a method of control for our button using Python. The right side of each button needs to be connected to GROUND to complete the circuit. We could plug each cable into a different GROUND pin on the Pi, but in order to keep things tidy and minimise the number of wires between the Pi and the breadboard, we've linked multiple buttons to a row on the breadboard and then wired that to a GROUND pin on the Pi. We've done this once for the top and again for the button of the board, so we're only using two GROUND pins in total.

In the image you can see the black and white GROUND cables at the top of the breadboard are connected to the Pi via a blue wire. It'd make sense to colour-coordinate and do the same thing at the bottom.



03 Testing the connection

Now that all the hardware is connected we're going to want to test the connection of each button in Python. First, Import the Button module, then create a variable for each one, with an assigned GPIO pin number. We've gone with BCM 2, 3, 4, 5 and 6 for the sake of simplicity for this project.

```
from gpiozero import Button
leftButton = Button(2)
```

04 Add a loop and print command

Now that you've got the variables set up correctly, create a simple while loop and add each button with

Retro Pi Phone

Retro accessories are all the rage – this project adds smartphone features to a 1960s rotary phone

Dan Aldred's Pi Phone is a 1960s rotary phone that uses a RasPi to enable you to dial for the latest headlines or send a tweet.

What inspired this retro project?

I was originally sourcing an old rotary dial phone to show my students what phones used to be like. Just before Christmas, I saw that a popular supermarket chain had released a new, modern version of the classic rotary dial telephone – except that they had replaced the dial with push buttons, and the main cord was connected to the router rather than the telephone exchange socket.

People below a certain age do not know how the classic rotary dial works. When dialling, they simply do not know how far to turn the rotate the dial, whether to take their finger out of the number hole and which way to turn the dial. Considering this, I was left with the idea of combining classic smartphone features, such as news, music and Twitter, with the traditional retro method of accessing information.

How difficult is it to connect a Pi to a rotary dial phone in terms of hardware?

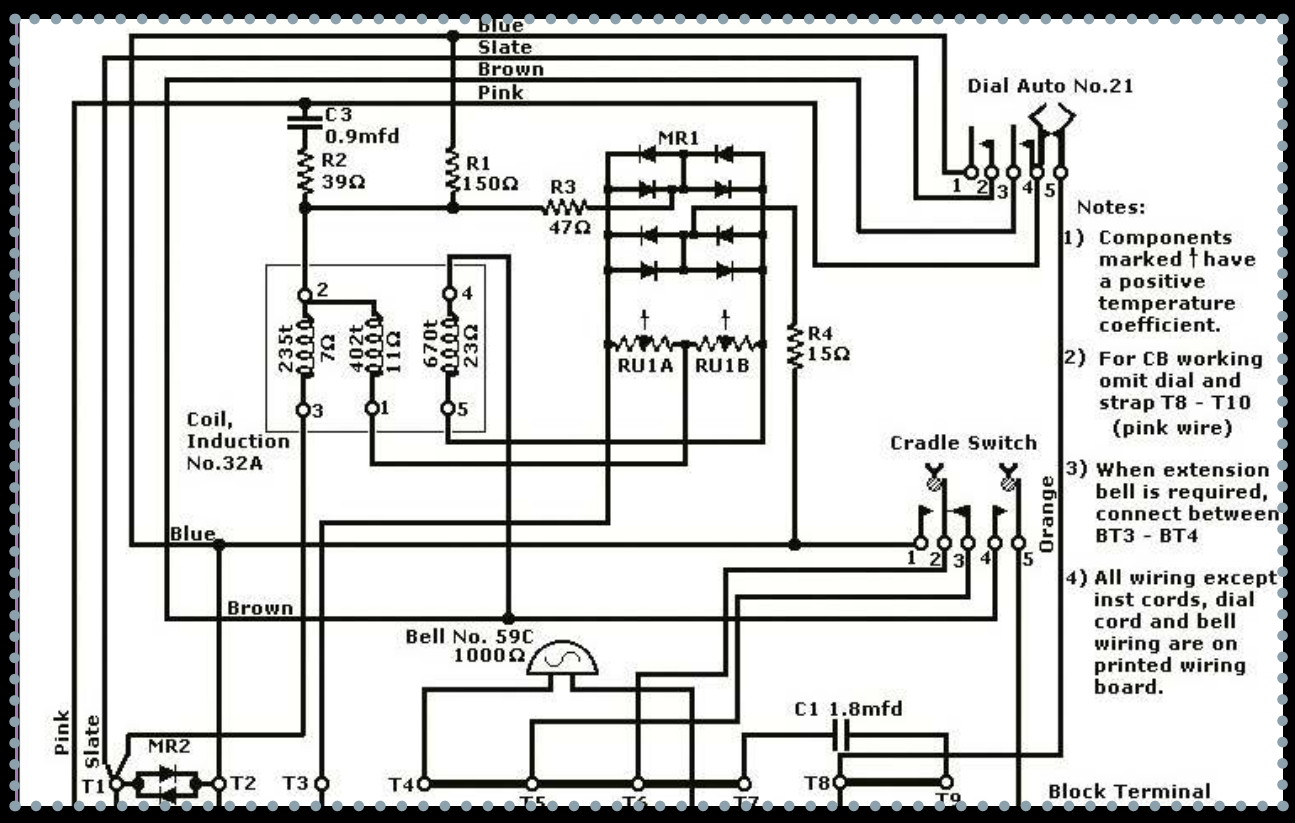
Basically, you use the Raspberry Pi to create a circuit between the GPIO pins and the rotary dial. You then use Python code to check for a change in state. As the rotary dial turns it makes contact, connecting and breaking a circuit. This change is then read via the GPIO pins and fed back into the program code.

The difficult part was locating a suitable wiring diagram for the model of phone that I was using. They all look the same, but the hardware inside is different depending on the age and model. www.britishtelephones.com contains a lot of advice

Dan Aldred Dan is the Head of ICT at a secondary school in North Yorkshire and a Raspberry Pi Certified Educator.

and wiring diagrams and was very useful. Once I had the correct diagram I could identify and locate the various coloured wires and hook them up from the rotary dial to the GPIO pins. As you dial the number, the 'pulse' value is returned and printed out to the Python shell. For example, dial 3 and it returns a value of 7, dial 8 and it returns a value of 17. A quick bit of maths and you can convert these pulse values into the actual numbers dialled: 7 minus 1, divided by 2, is 3; and 17-1 divided by 2 is 8. Once you know which number has been dialled you can assign a function to it. For more details and an in-depth look at the wiring see https://youtu.be/JChn7tV_qBk.

The main element to focus on is getting the rotary dial talking to your RasPi. Once this is established, you can customise the functions. Also consider which model Pi to use. Originally, I had intended to embed a Pi Zero W due to its small size. However, as I was developing I soon realised that I required USB ports and a fast processing speed. I had to strip out some of the telephone's original components to make room to house a Raspberry Pi 3 B+.



Left Dan says that getting the rotary dial connected to the Raspberry Pi 3 was the most complex part of the project, as he had to find the phone model and identify what each of the wires did. Fortunately, he found schematics on the web. Basically, it's the gates in the top right-hand corner of this diagram.

Can you tell us about some of the Python scripting you used?

Once you can read the pulse from the rotary dial you can assign actions to each number. This is a list of 10 conditionals. If number 4 is dialled, then play an MP3; if 0 is dialled, then read out the saved voice message.

The speech recognition is a powered by Google's standard recognition system and works very well, as long as the microphone is of a good quality. It records your voice, sends the file to Google where it is converted into a text string, and then sent back to the Python program. I also used the classic e-speak to read out all text.

The SMS system is powered by Twilio and uses an API and a unique telephone number to check for text messages. When an SMS is received, Twilio, the Pi and the Python code respond by triggering the ring-ring sound, and then converts and stores the text content of the message into speech.

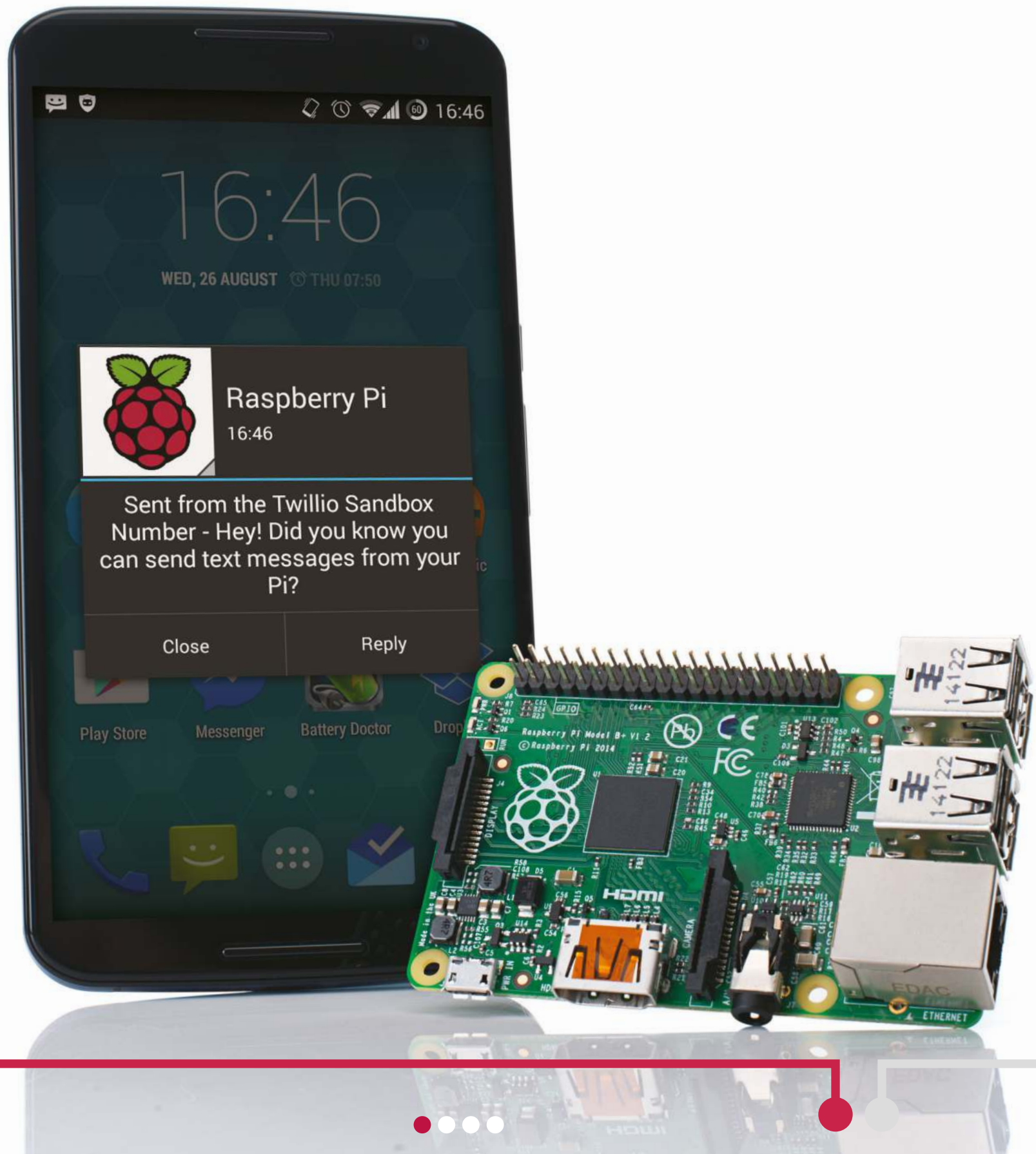
> "Once you can read the pulse from the rotary dial you can assign actions to each number"

Is there a feature you really want to see in the next Raspberry Pi, or perhaps for an add-on board if that's a better fit?

I would like to see some sort of scanner or RFID sender/transmitter that fits directly onto a Pi Zero. With that sort of setup you could create your own remote key or trigger notifications when certain conditions are met.

Send an SMS from your Raspberry Pi

Create a program that combines Twilio and simple Python code to enable you to send an SMS from your Pi to a mobile phone



Text messaging, has become a staple of everyday communication. What began life as a 40 pence message service is now offered as an unlimited service used worldwide. Twilio, a cloud communications company, enables you to send SMS messages for free from your Raspberry Pi to a mobile phone using just six lines of code.

THE PROJECT ESSENTIALS

Raspberry Pi

Twilio account

01 Set up your Twilio account

The first step of this project is to register for a Twilio account and Twilio number. This is free and will enable you to send an SMS to a registered, verified phone. Once signed up, you will receive a verification code via SMS to the registered phone. When prompted, enter this onto the Twilio site to authenticate your account and phone. Go to twilio.com/try-twilio and create your account now.

02 Register numbers

Your Twilio account is just a trial account unless you pay the upgrade fee, which means you can only send and receive communications from a validated phone number. Enter the phone number of the contact who you want to verify, ensuring that you select the correct country code. Twilio will text you a verification code and you will need to enter it into the website form and press submit.

03 The dashboard

Once registered and logged in on Twilio, visit the dashboard page, which will display your AccountSid and your Auth Token. These are both required to use the Twilio REST. Keep these secure and private, but be sure to make a note of them as you will need them for your Python program later.

04 Install the software

Now boot up your Raspberry Pi and connect it to the internet. Before you install the Twilio software, it is worth updating and upgrading your Pi. In the LX Terminal, type sudo apt-get update, then sudo apt-get upgrade. Once complete, type sudo easy_install twilio or sudo pip install twilio to install the software. (If you need to install pip, type sudo apt-get install python-pip python-dev, press Enter, then type sudo pip install -U pip.)

05 Twilio authentication

Now you are ready to create the SMS program that will send the text message to your mobile phone. Open your Python editor and import the Twilio REST libraries (line one, below). Next, add your AccountSid and Auth Token, replacing the X with yours, as you will find on your dashboard:

```
from twilio.rest import TwilioRestClient
account_sid = “XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX”
# Enter Yours
auth_token = “XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX”
# Enter Yours
client = TwilioRestClient(account_sid, auth_token)
```

06 Create your message

You will probably want to be able to change your text messages rather than send the same one. Create a new variable in your program called message. This will prompt you to enter the phrase that you want to send to the mobile phone. When the program runs, this is the

message that will be sent:

```
message = raw_input("Please enter your message")
```

07 Add your numbers

To send the message, you need to add the code line below and your two phone numbers. The first number is your mobile phone number, which is registered and validated with Twilio (Step 2). The second number is your Twilio account number, which can be retrieved from your dashboard page under 'Call the Sandbox number'. Change the Sandbox number to your country location and remember to add the international country code.

```
message = client.messages.create(to="+44YOURMOBNUMBER", from_="+44YOURTWILIONUMBER", body=message)
```

08 Send the message

Now send your message. The code below is not required, but is useful to indicate your message has been sent. Add the lines and save your program. Ensure your Raspberry Pi is connected to the internet and that your mobile is on, then run your program. You have just texted from your Raspberry Pi.

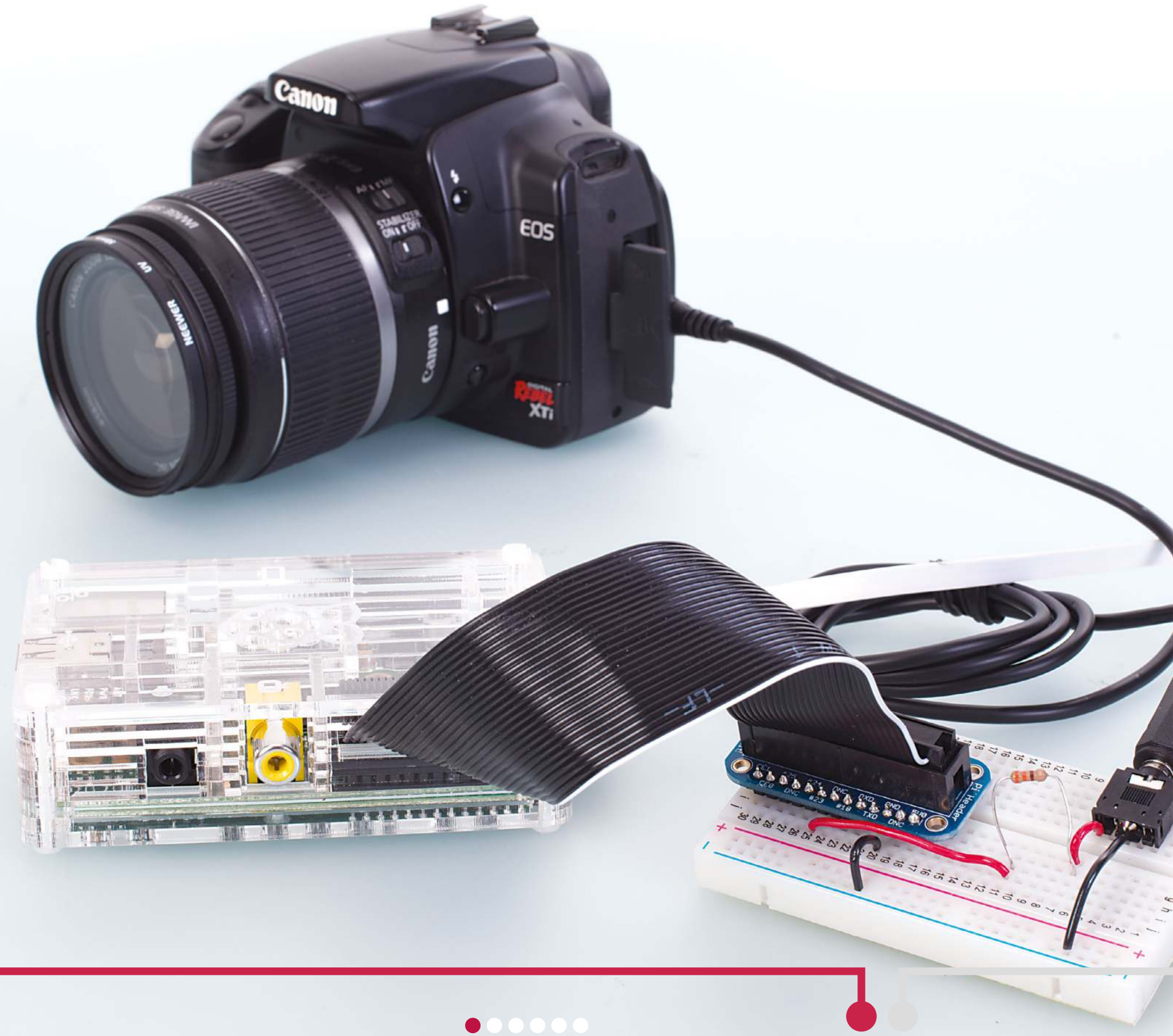
```
print message.sid
print "Your message is being sent"
print "Check your phone!"
```

09 Other API and codes

You can also create other comms programs like phone calls and retrieving data. The API here also complements a range of programming languages, including Ruby, PHP, Java and Node.js (twilio.com/api) to name a few.

Time-lapse camera trigger

Make shooting time-lapse video with your DSLR camera a cinch with our expert advice



You'd be forgiven for thinking that creating mesmerising time-lapse videos like those of Vincent Laforet (www.laforetvisuals.com) or John Eklund (www.theartoftimelapse.com) might be out of reach of the Average Joe. With the help of the Raspberry Pi and a sprinkling of Python code, though, that's no longer the case. In this guide we're going to trigger our DSLR camera to create pixel-perfect time-lapse imagery...

01 Set up the Raspberry Pi

For this tutorial we're assuming you're using a recent build of Raspbian. With the Raspberry Pi set up with a keyboard, mouse and monitor, open the terminal and type:

```
sudo apt-get update
```

02 Install the RPi.GPIO library

Next we want to make sure your development environment is set up. Follow these steps to make sure you're all set.

In the terminal, type:

```
suda atp-get install python-dev
sudo apt-get install python-rpi.gpio
```

03 Set up the Pi Cobbler

For this tutorial we've used a cheap prototyping breadboard and an Adafruit Pi Cobbler to give us easy access to the Raspberry Pi's GPIO pins. As you can see from the picture, the Cobbler straddles the centre-point of the breadboard and a ribbon cable connects the two.

THE PROJECT ESSENTIALS

Latest Raspbian Image

www.raspberrypi.org/downloads

Breadboard

Connectors

Jumper wire

DSLR camera

Compatible shutter cable

04 Configure the breadboard

For the Raspberry Pi's GPIO to control the camera, we need to create a circuit between a pin on the GPIO (in this case pin 23 on the Cobbler – but it's actually physical pin 16) and the pin that connects to the 'head' or 'tip' of the camera cable that activates the shutter when connected. The base of the connector cable is always ground, so make sure you ground the 'GND' pin on the Cobbler and the middle pin on the audio jack. With the circuit complete, we can focus on the code.

05 The Time-lapse Photography Tool

We've created a small 55-line Python utility called The Linux User Time-lapse Photography Tool, which asks the user to input how many shots they'd like to take and the frequency they'd like them taken. It then takes that information and uses it in a For loop to activate the shutter using GPIO pin 16 . If you'd like to use the project 'in the field' we'd recommend using the Android app ConnectBot to SSH into your RasPi for input and feedback. Don't forget to start your script with sudo python time_lapse_camera.py

06 Creating a video

With your camera packed with images, we need to collect and output them as a video file. While it's possible on the Pi, copy them to an easily accessible folder on a separate Linux PC to make it much faster. We're going to use FFmpeg. With the terminal open in the folder where your images are stored, type: ffmpeg -f image2 -i image%04d.jpg -vcodec libx264 -b 800k video.avi. This assumes you have libx264 installed on your machine and the 'image%o4d.jpg' assumes the file format and the number of digits it's dealing with (in this case: 'picture0001.jpg').

Full code listing

```
import RPi.GPIO as GPIO
import time

print '\nWelcome to the Complete Manual Time-lapse Tool.'
print "Just tell us how many shots you'd like to take and the interval between them.\n"
print "Try googling 'time-lapse interval calc' if you need help deciding.\n"


def main():
    shots = raw_input('How many shots would you like to take?\n ->')
    interval = raw_input('How frequently do you want to take them (in seconds)?\n ->')

    if shots.isdigit() and interval.isdigit():
        shots = int(shots)
        interval = int(interval)

        print "You'll be shooting for %d minutes.\n" % (shots * interval / 60)
        answer = raw_input('Are you ready to proceed?(yes/ no):')
        confirm = answer.lower() in ['yes', 'y']

        if confirm:
            GPIO.setmode(GPIO.BOARD)
            GPIO.setup(16, GPIO.OUT)
            taken = 1
            print
            print 'Starting a run of %d shots' % (shots)
```

```
            for i in range(0, shots):
                print
                print 'Shot %d of %d' % (taken, shots)
                taken +=1
                GPIO.output(16, GPIO.HIGH)
                time.sleep(0.5)
                GPIO.output(16, GPIO.LOW)
                time.sleep(interval)
            GPIO.cleanup()
        else:
            print "Let's try again (or press Ctrl + C to quit):\n"
            main()
    else:
        print "Oops - You can only enter numbers. Let's try again:\n"
        main()

    print
    print 'Thanks for using the Complete Manual Time-lapse Tool!\n'
    again = raw_input('Would you like to do another time-lapse? (yes/no):\n -> ')
    proceed = again.lower() in ['yes', 'y']

    if proceed:
        main()
    else:
        print '\nSee you next time!\n'
        quit()

if __name__ == '__main__':
    main()
```

Manual focus
We won't be controlling the autofocus with our Python app, so set the focus to manual and select your camera settings in advance of the shoot

2.5mm to 3.5mm
We're using a cheap Canon EOS DSLR, so to trigger the shutter with the Raspberry Pi, all we need is a simple 2.5mm to 3.5mm cable

Pi Cobbler
We're using the Pi Cobbler as a breakout for the Pi's GPIO pins, making the build process easier (though it's not required)

Set up a Jupyter server on your Pi

Jupyter provides a great interface that you can use through your Raspberry Pi for collaborative computational or data science projects

The Raspberry Pi has found a cosy niche in several projects where it's used as the computational core. This led to the Pi Foundation releasing the Compute Module, designed specifically for this use case. This month, we will build on this idea by looking at the Python project, Jupyter. Through its web-based interface, Jupyter provides a great front-end to the computational resources you have available through your Raspberry Pi. If you are using the stock Python installation, you can install Jupyter with the following commands.

```
sudo apt-get install python-dev
sudo pip install --upgrade pip
sudo pip install jupyter
```

If you installed Berryconda, the Anaconda port for the Raspberry Pi, you can install it from there as well.

Once you have it installed on your system, you will need to start it up. The assumption we'll be making is

that your Raspberry Pi is headless, somewhere on the network, and that you are working on it over SSH. In this case, we'll also assume that you are using some form a terminal virtualisation, such as screen or tmux. Install tmux with:

```
sudo apt-get install tmux
```

Then start it with the command tmux. The advantage here is that you can easily log back into the Pi and reattach to the tmux instance in order to see what is happening with your Jupyter process. There are a large number of options available when starting up Jupyter, which can be expressed as command-line options. That means that you need to remember what options are being used, which can be difficult, so instead you can put all of these options in a configuration file that Jupyter can use. To create a configuration file with the default options, use the following command.

```
jupyter notebook
--generate-config
```

This will create a new file located at ~/.jupyter/jupyter_notebook_config.py. All of the available options should be included, commented out, with their default values listed. Going down the list, the first option you'll want to edit is c.NotebookApp.ip. The default value has the Jupyter notebook only listening to incoming connections from the same machine. In order to allow connections from other machines, you will want to uncomment this line and change the value to be '*'. This tells Jupyter to allow connections coming in on any of the network

Why Python?

It's the official language of the Raspberry Pi. Read the docs at https://www.python.org/doc/

interfaces. This means that you are going to be open to potential intrusions, so you will need to be more aware of security issues. The next option is c.NotebookApp.notebook_dir. This is the directory where Jupyter defaults to reading and writing notebook files. For this article, we'll assume that you have created a subdirectory named notebooks in your home directory. This option line will then look like the following:

```
c.NotebookApp.notebook_dir = 'notebooks'
```

By default, Jupyter wants to open a browser immediately upon starting up, but this is not what you'll want to have happen on a remote machine. So, uncomment the option c.NotebookApp.open_browser and set its value to False. The last most common option that will need to be changed is the listening port. Labelled as c.NotebookApp.port, it defaults to 8888. If you have other services running on your Pi, this port may already be in use. Once all this is done, you can start up Jupyter with:

```
jupyter notebook
```

Depending on the version of Jupyter, you may see that there is a token set within the log output. This is a security measure, and you will need to include that token when you enter the URL to your Raspberry Pi in order to be allowed to connect to the server.

Now that you have a Jupyter server running, what can you do with it? When you first load the dashboard, you will see a list of any existing folders and notebook files. If it is your first time, it should be empty. In the top-right corner, there is a new button which will allow you

to create a new notebook. Depending on what versions of Python you have installed, you may have an option of whether to create a notebook based on Python 2 or Python 3. There are also other engines possible, such as R and Julia, but we won't be covering them here. Creating the new notebook will create a new browser window where you can start doing your Python work. The notebook is cell based. Each cell can be thought of as a separate block of code, which is executed as a single unit. This means you can edit an entire cell's worth of code before executing it. Execution happens when you either press Shift+Enter, or you can click on the 'Execute Cell' icon at the top of the notebook. You can also go back to a previous cell, edit it and rerun it. But this brings up an idea that needs to be kept in mind. The Jupyter notebook engine has a chronologically based state. This is similar to commercial mathematical software, such as Maple and Mathematica. This means that if your cell sets a value to a variable and then gets rerun, it may change the value within that variable the next time it accessed. The input and output is managed in the same way as in IPython, with 'In' and 'Out' labels. Actually, the default engine is IPython, so you have access to all of the familiar tools available in IPython. For example, you can access the last three values returned with the special variables '_', '__' and '___'. You can also use the IPython magics to interact with the engine. For example, if you have a function named do_work(), you could time it using the following code.

```
%timeit do_work()
```

You will get output that resembles the following:

```
7.82 ms ± 33.6 µs per loop (mean ± std. dev.
of 7 runs, 100 loops each)
```

If, instead of a single of line code, you can time an entire cell of code by starting it with the line %%timeit. When the notebook reaches a certain size, you may not be entirely sure just what state the IPython engine is in. In these cases, you can click on the menu item Kernel > Restart & Run All. This clears all output and recalculates all of the cells in the notebook in the order that they are written in. This is a good way to ensure that you are in the state that you think you are in.

Jupyter is also very well situated for literate coding, allowing your documentation to live embedded with the computations being run. You can change the type of cell to be Markdown text by clicking the menu item Cell > Cell Type > Markdown. You can then write documentation, using Markdown to handle the formatting. Just as with computational cells, you need to run it in order to have the formatting processed. This embedded documentation means that you can easily export an entire document that contains both theory, calculation, analysis and results as a single object. To make this easier, there are several formats that you can download your notebook as. They are all located under the menu item File > Download as. You can get a fully rendered HTML page of your notebook, suitable for blogs or websites. If you wanted to generate a scientific document, you can download the notebook as a LaTeX file which can be included within your submission. If you have the software needed on your Raspberry Pi, you can also download a PDF version generated from the LaTeX version. Before you do, don't forget to retitle your notebook by clicking the menu item File > Rename...,

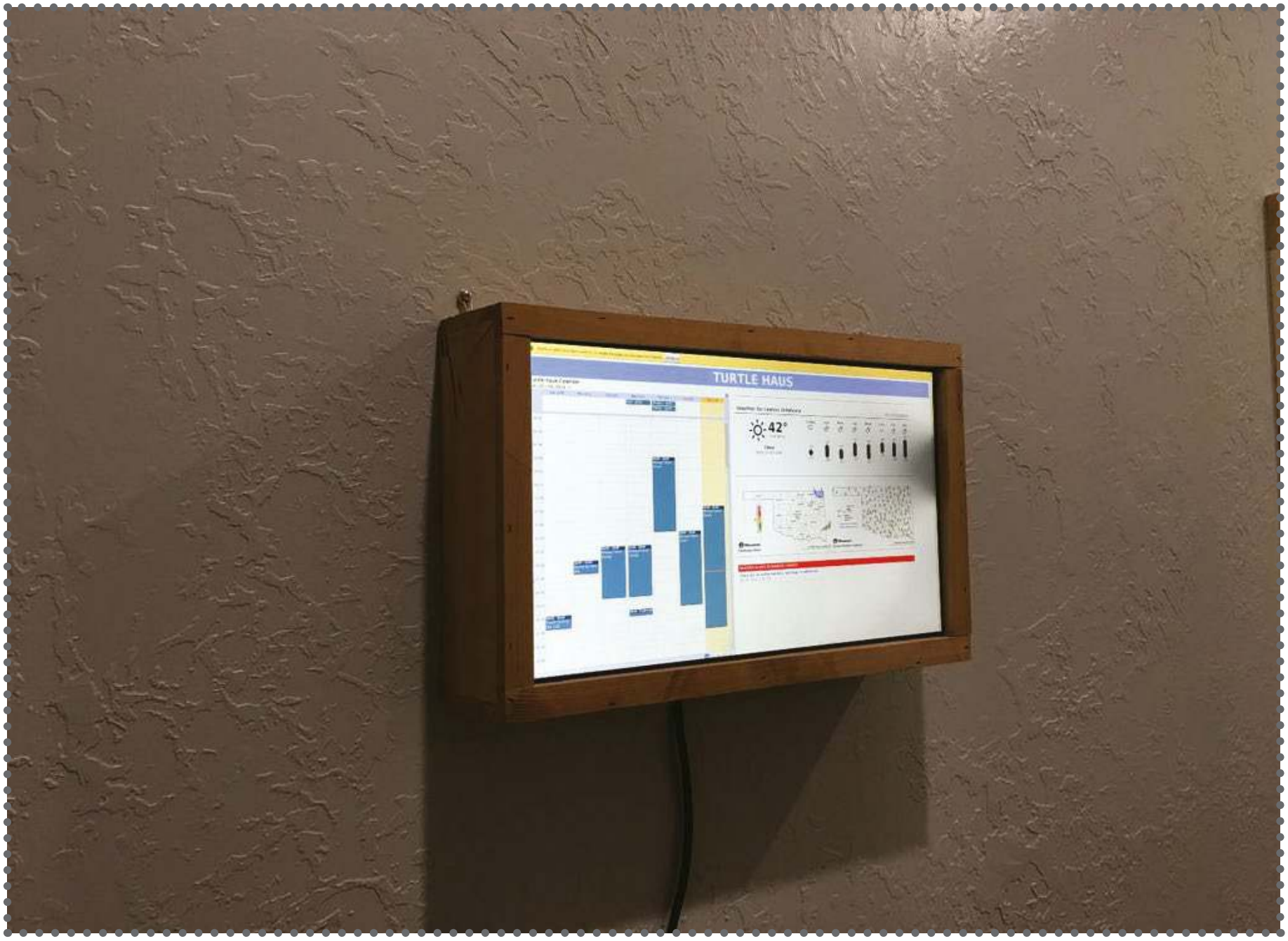
otherwise you will end up with a large number of notebooks titled as 'Untitled'. If you are using Jupyter to build more educational documents, you may want to reset your notebook. Clicking the menu item Kernel > Restart & Clear Output will give you a fresh notebook with all of the output removed.

When you need to shut down the Jupyter server, you can SSH in to your Raspberry Pi and reattach to the tmux instance. From here, you can press Ctrl+C to tell Jupyter to shut down all of the kernels cleanly. It will only give you five seconds to confirm, so don't go away too quickly. Now you have a tool to put those loose Raspberry Pis on your network to work, crunching numbers or processing data.

Next issue

Get inspired **Expert advice** **Easy-to-follow guides**

"Build a smart calendar"



Get this issue's source code at:

www.linuxuser.co.uk/raspicode